DOCUMENT RESUME

ED 422 925 IR 057 083

AUTHOR Urbaczewski, Andrew; Urbaczewski, Lise

TITLE Beyond Course Availability: An Investigation into Order and Concurrency Effects of Undergraduate Programming Courses on Learning.

PUB DATE 1997-00-00

NOTE 7p.; In: Proceedings of the International Academy for Information Management Annual Conference (12th, Atlanta, GA, December 12-14, 1997); see IR 057 067.

PUB TYPE Reports - Research (143) -- Speeches/Meeting Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.

DESCRIPTORS Academic Achievement; Higher Education; Information Science Education; Instructional Effectiveness; Introductory Courses; *Programming; *Programming Languages; Student Attitudes; Student Surveys; Teaching Methods

ABSTRACT

The objective of this study was to find the answers to two primary research questions: "Do students learn programming languages better when they are offered in a particular order, such as 4th generation languages before 3rd generation languages?"; and "Do students learn programming languages better when they are taken in separate semesters as opposed to simultaneously?" Students from nine introductory programming classes over two semesters at a large Midwestern university were used as subjects for this experiment; 275 students responded to a survey at the end of the semester. Subject responses were divided into three groups, depending on the class being rated. These classes were: introduction to Visual Programming, introduction to COBOL programming, and introduction to C programming. To test hypotheses, linear regression was used, running the data against two separate dependent variables, grade and comfort factor. Mixed results were found for the different types of classes. (AEF)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED 422 925

BEYOND COURSE AVAILABILITY: AN INVESTIGATION INTO ORDER AND CONCURRENCY EFFECTS OF UNDERGRADUATE PROGRAMMING COURSES ON LEARNING

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☐ This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

Andrew Urbaczewski
Indiana University

Lise Urbaczewski
Indiana University

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
T. Case
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

INTRODUCTION

The process of undergraduate MIS education is constantly evolving. As educators, we find ourselves in a constant state of curriculum redesign, often to meet the demands of the various recruiters that visit our campuses each Fall looking for new talent in the expanding field. Quite often, these new technologies are programming languages. It is common today to hear recruiters ask for students skilled in C++,Visual Basic, Powerbuilder, or Java before entering the job market. In accordance with Association for Computing Machinery (ACM) curriculum design guidelines (ACM 1991), we incorporate these technologies into our curriculums, keeping the content as current as possible. However, in the rush to design our curriculums to give students maximum exposure to required technologies, perhaps we overlooked factors to maximize learning efficiency in our students.

A debate exists today over the proper method of programmer instruction. Research has been conducted in the past to find the optimal sequence for offering programming instruction. Veteran programmers learned the older second and third generation languages (2GLs and 3GLs) before they learned 4GLs because no 4GLs existed. Students now are often afforded the opportunity to learn a 4GL or object-oriented programming language without ever learning a 3GL. Empirical research in the past has found mixed results (Manns and Carlson 1992, Rosson and Alpert 1990), while authors have made claims for first learning the object oriented language (Currid 1992) and for first learning the 3GL (Powell 1997).

Moreover, students often find themselves in a crunch to get registered for the required courses. With the undergraduate MIS major's increasing popularity, it is getting harder for the students to arrange their schedules' optimally. It is not uncommon to find students taking two or three different programming languages during the same semester. The authors find it difficult to believe that a student can perform optimally under these conditions, as this would be like a student trying to learn two spoken foreign languages, like French and German, during the same semester.

Given the apparent mentioned conflicts, the objective of our study was to find the answers to 2 primary research questions:

1) Do students learn languages better when they are offered in a particular order, such as 4th Generation Languages (4GL) before 3GL's or vice versa or is there no effect?

2) Do students learn programming languages better when they are taken in separate semesters as opposed to simultaneously?

In an attempt to find the answers to these questions, we decided to ask the students themselves. In the classroom, programming instructors hear a variety of complaints from students regarding their difficulty or inability to

IR057083

learn the language at hand. We divide these comments in two groups. The first group is from students who have learned a prior language and are having difficulty learning a second language. Often they are attempting to learn two programming languages at the same time. This is the group we call "But there's an easier way....." The other group are students that have never had any programming courses before. They tend to report in the classroom that they feel inferior to other students in the class who may know more about computers or be more experienced with programming. They are worried that their objective performance in the class will suffer because they are being compared to this other group of students. This group we call "It's not fair....."

Based on experiences noted in the classroom, we formulate the following hypotheses:

H1a. Students that have a prior experience with a programming language will have higher grades in an introductory course in that language than those who have no prior experience with that programming language.

H1b. Students that have taken any programming class will have better grades in a course than those who have not taken a programming class.

H1c. Students who are taking more than one programming language simultaneously will have worse grades than those who have taken the same programming languages in any non-simultaneous order.

We use grades in H1 because they are intended to be an objective measure of performance. However, there are often other factors than mastery of the skill which figure into the awarding of grades. These often include attendance, year of progression through school, and some factor of general intelligence. Therefore, we also propose:

H2a. Students who attend class more regularly will have higher grades in programming courses than those who do not attend class regularly.

H2b. Students who have higher grade point averages will perform better in a programming course than those who have lower grade point averages.

H2c. Students who have progressed further in school will have higher grades in programming courses than those who have progressed less in school.

We would still like to get another dependent measure of mastery of the skill. While objective measures are important in science, a subjective measure of mastery may also be important. Students may make a high grade in a course but not have any mastery of the material, and vice versa. Therefore we propose level of comfort as a dependent measure for student mastery of programming material.

H3a. Students that have a prior experience with a programming language will feel more comfortable with that language than those who have no prior experience with that programming language.

H3b. Students that have taken any programming class will feel more comfortable with the language than those who have not taken a programming class.

H3c. Students who are taking more than one programming language simultaneously will feel less comfortable with the languages than those who have taken the same programming languages in any non-simultaneous order.

METHODOLOGY

Students from nine introductory programming classes over two semesters at a large Midwestern university were used as subjects for this experiment. 275 students responded to a survey at the end of the semester. This was done without compensation to the students, requiring a few minutes of their time at the beginning of the class. One response was determined to be unusable and it was discarded. Subjects were also asked that if they had completed this survey in another class to indicate this at the top of the page, and 19 surveys were eliminated through this method.

The survey asked students to respond to several items concerning their academic performance and relative comfort with the language. Students responded anonymously to remove threats to internal validity (Campbell and Stanley 1963). These items covered objective performance levels, such as grading, and subjective performance, including comfort with the language and desire to learn more about the language. Subjects reported not only on the class they were currently finishing but also on all other programming classes they had taken for a grade at the collegiate level or above. This was done to capture data about all their classes for determination of ordering effects.

Subject responses were divided into three groups, depending on the class being rated. These classes were introduction to Visual Programming, introduction to COBOL programming, and introduction to C programming. This of course meant that the same subject may have been in each of the three pools if she or he had taken all of the courses being examined. These groups were analyzed individually and then compared to the others for significance. Linear regression was used with the same covariates against two different dependent variables, grade in the course and comfort level with the language. Reported prior experience was also captured as a Likert scale and used as a covariate.

RESULTS AND INTERPRETATIONS

To test our hypotheses, we used linear regression, running the data against two separate dependent variables, grade and comfort factor. As noted above, grade was the self reported letter grade for the student for prior classes and expected grade for the current class. This was then converted into the numeric equivalents at that university. The data set was also divided into three partitions, depending on the class being rated. Since attendance was expected to have only a significant influence on grade rather than comfort level, they were only suggested as hypotheses when grade was the dependent variable. We inserted the variables for H2 into those regressions anyway for the reader's benefit.

1. Dependent Variable = Grade

A. Class Rated = Introduction to Visual Programming using Visual Basic

	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	S.E.	Beta		
(Constant)	.560	.537		1.043	.299
W-COBOL	-.359	.420	-.062	-.856	.393
W-C	.176	.119	.115	1.483	.140
H-COBOL	<.001	.126	.069	.750	.454
H-C	<.001	.121	.075	.793	.429
Attendance	.151	.050	.223	3.053	.003
PriorExp	<.001	.035	.213	2.822	.005
GPA	.397	.123	.234	3.242	.001
YrInColl	<.001	.065	.048	.615	.539

B. Class Rated = Introduction to COBOL programming

	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	S.E.	Beta		
(Constant)	1.322	1.006		1.315	.195
W-VB	.195	.464	.050	.421	.675
W-C	-.164	.220	-.092	-.746	.459
H-VB	-.281	.374	-.099	-.750	.457
H-C	-.159	.235	-.095	-.677	.501
Attendance	<.001	.098	-.066	-.525	.602
PriorExp	<.001	.089	-.092	-.731	.468
GPA	1.183	.243	.605	4.876	.000
YrInColl	-.315	.167	-.246	-1.890	.065

C. Class Rated = Introduction to C programming

	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	S.E.	Beta		
(Constant)	.549	.447		1.229	.221
W-VB	.276	.105	.170	2.635	.009
W-COBOL	-.297	.162	-.117	-1.834	.068
H-VB	<.001	.128	.046	.743	.458
H-COBOL	<.001	.103	.013	.196	.845
Attendance	<.001	.045	.112	1.724	.086
PriorExp	<.001	.024	.185	2.901	.004
GPA	.753	.102	.484	7.356	.000
YrInColl	<.001	.056	-.023	-.334	.739

It is interesting that we find different results for the different types of classes. This would support the mixed results that we have seen before. For the visual programming courses, we find grades are in no way related to any other programming language work. Attendance, Prior experience, and GPA are the only significant factors affecting grade, supporting **H1a, H2a,** and **H2b**. We fail to reject the null hypothesis for **H1b, H1c**, and **H2c**.

With the course in COBOL, however, we get different results. The only significant predictor of grade is GPA, supporting **H2b**. It is interesting that **H1a** is not supported, perhaps giving credibility to the notion that grades and mastery are not perfectly related. Perhaps even more significant is that year in school is marginally significant (p=.065), in the opposite direction, suggesting that students who are further along in school will perform worse in COBOL. This may be evidence for the term "senioritis", suggesting that students may slack off on their studies as they get closer to graduation.

Finally, the Introduction to C programming course gives even different results. **H1a** is supported, confirming that those who have prior experience will earn a better grade. Taking the course with the visual programming course improves their grade in C, completely opposite of **H1c**. However, taking the course with COBOL is marginally significant (p=.068), supporting **H1c**. This is especially interesting given that C is more closely related to COBOL than it is related to Visual Basic. **H2b** (GPA) is also supported, and there is mild support (p=.086) for attendance improving grades.

2. Dependent Variable = Comfort Level

A. Class Rated = Introduction to Visual Programming using Visual Basic

	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	S.E.	Beta		
(Constant)	3.223	1.038		3.106	.002
W-Cobol	.983	.664	.107	1.481	.140
W-C	.536	.223	.188	2.407	.017
H-Cobol	.387	.244	.146	1.586	.115
H-C	.169	.234	.069	.721	.472
Attendance	<.001	.096	.052	.705	.482
PriorExp	.158	.067	.178	2.350	.020
GPA	<.001	.237	.016	.220	.826
YrInColl	.142	.125	.089	1.142	.255

B. Class Rated = Introduction to COBOL programming

	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	S.E.	Beta		
(Constant)	4.572	2.074		2.204	.032
W-VB	.479	.798	.081	.601	.551
W-C	.234	.458	.073	.510	.612
H-VB	-.281	.779	-.055	-.360	.720
H-C	.632	.461	.218	1.369	.177
Attendance	.245	.200	.175	1.225	.226
PriorExp	<.001	.186	-.011	-.077	.939
GPA	<.001	.498	-.015	-.103	.919
YrInColl	-.346	.347	-.150	-.998	.323

C. Class Rated = Introduction to C programming

	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	S.E.	Beta		
(Constant)	.560	.537		1.043	.299
W-VB	.674	.194	.231	3.476	.001
W-Cobol	-.725	.325	-.148	-2.232	.027
H-VB	.686	.281	.160	2.442	.015
H-Cobol	<.001	.221	.017	.244	.807
Attendance	<.001	.097	.028	.406	.685
PriorExp	.183	.053	.230	3.449	.001
GPA	.299	.212	.096	1.408	.161
YrInColl	-.103	.112	-.065	-.927	.355

Again in examining the Comfort variable, we get mixed results. These results also tend to be different from those using grade as a dependent variable, lending more credibility to the notion of grades and mastery being mildly correlated. In the Visual Programming class, **H3a** is supported, which may suggest that one semester of a visual programming course is not enough to bring novices on a par with more experienced users. We were again surprised to find that **H3c** was supported in the opposite direction than we suspected, as increased comfort level was reported from those who took the two classes together.

In the COBOL course, we find nothing significant. None of the variables we captured were predictors of a person's comfort level with COBOL.

The lack of support for many variables in the COBOL course in both situations may be due to a lack of power. There were far less students who had taken the COBOL course (56) than the C course (191) or the Visual Programming course (173).

Finally, in the C course, we have a third set of results. In line with the Visual Programming course, students reported a higher comfort level if they were taking the course concurrently with visual programming, providing opposite support for **H3c**. The more expected result was found from students who were taking C with the COBOL class, supporting **H3c. H3b** was also supported with the Visual Programming course, but it was not with the COBOL course. This was the only instance where either **H1b** or **H3b** was significant. Prior experience was also significant, supporting **H3a**.

CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

As we noted in our results section, the findings we had with the data were very mixed. No single hypothesis was supported across all classes when Comfort was the dependent variable, and only GPA was supported consistently as a predictor of Grade. These mixed results are not necessarily bad, however. They help us in curriculum design, which was the goal of this paper.

The Introduction to COBOL course was the one which had the least predictors for either dependent variable. GPA was the only predictor of Grade, and there were no predictors of comfort level. This can be interpreted to mean that nothing gives anyone an undue advantage in the COBOL course. Beginners are just as likely to do well in the course as are experts. Thus we can suggest Introduction to COBOL as the first programming course in the programming sequence.

Secondly, we noticed that students seemed to perform better in the C course when they had already taken or were currently taking the visual programming course. This did not hold true in reverse. Thus we can suggest that students should take visual programming first, and then a course in COBOL.

FIGURE 1

Suggested Order of Courses:

1. COBOL
2. Visual Programming
3. C

More research should be done to try to discover the reasons for the contrasting results. For example, it is not clear to the authors why students seem to perform better when taking C concurrently with visual programming but worse when taking it with COBOL programming. We cannot explain these effects and would like to try to investigate them further. Exploration into student experiences with taking concurrent programming languages would also help, at least on a exploratory level. Perhaps comparing them to students who have attempted to learn two foreign languages simultaneously would help discover learning and memorization patterns.

Moreover, we are unsure how this research applies outside university settings. It would be interesting to survey professional programmers in the same manner. Professional training courses could be used to find programmers of different ability and style.

Overall, this research is of interest to researchers in one of our forms of patronage to society, educating students. We owe it our students (and often our taxpayers) to help them as much as possible in the education process. Finding the optimal way to deliver that education is part of our responsibility.

PARTIAL REFERENCES

ACM Curricula Recommendations Volume II: Information Systems, Association for Computing Machinery, 1991.

Currid, C. "IS Curriculum Continues to Evolve at the Corporate Level", *Infoworld*, June 8, 1992, p. S64.

Manns, M. and D. Carlson, "Retraining Procedural Programmers: A Case Against "Unlearning", Object Oriented Systems and Language *Applications Conference, 1992, pp. 131-133.*

Powell, A. "Programming Course Sequence and Prior Knowledge of Programming Languages: Do they Affect Students' Grades?" *Proceedings of the AIS Americas Conference on Information Systems, 1997.*

Rosson, M. and S. Alpert. "The Cognitive Consequences of Object-Oriented Design", *Human Computer Interaction*, 1990, pp. 360-378.

Webster's New Collegiate Dictionary, 1993.